# Logic and Computation – CS 2800 Fall 2019

## Lecture 5

## ACL2s continued

## Conses, lists, true lists

Stavros Tripakis

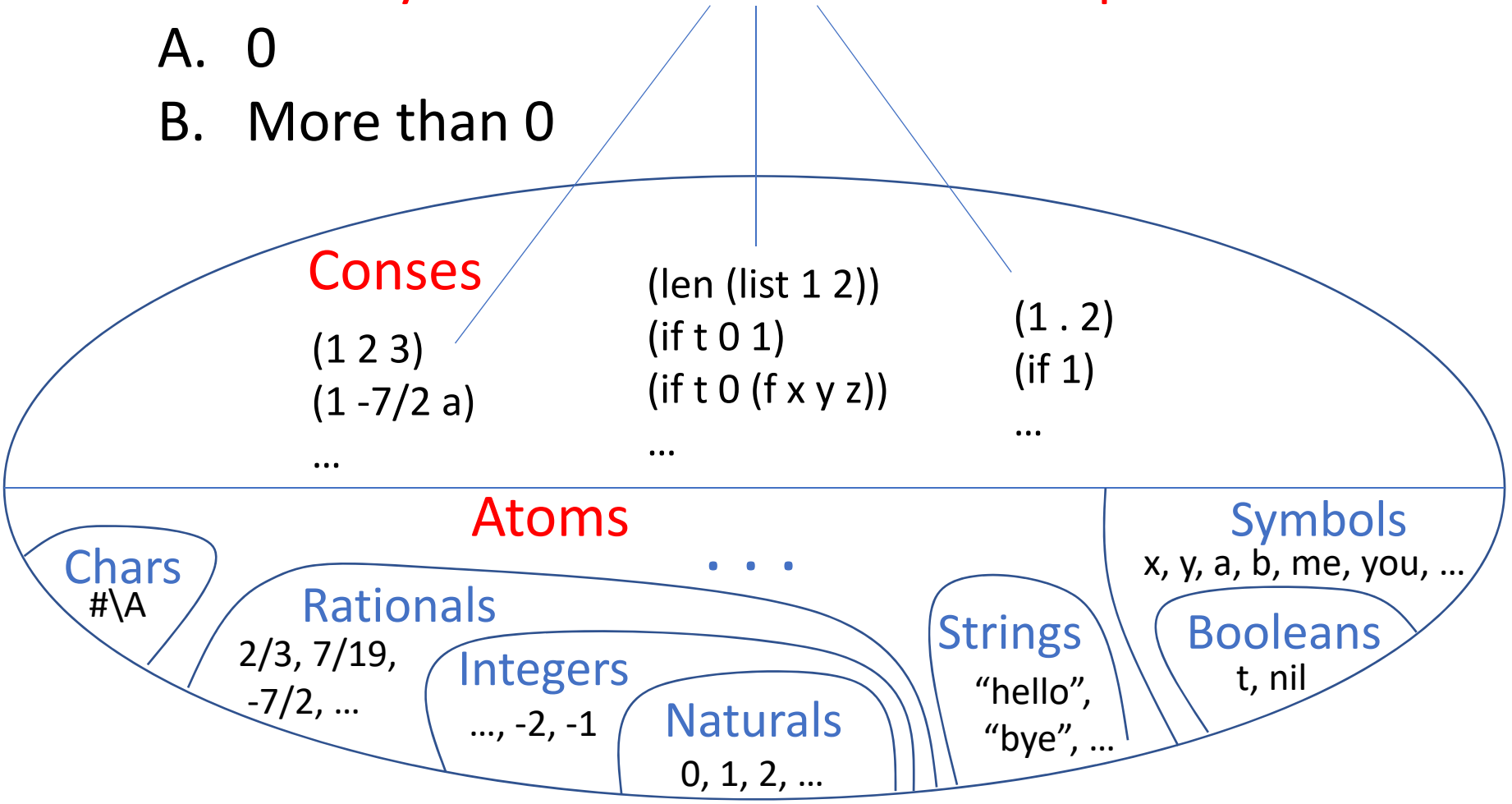

Northeastern University
Khoury College of
Computer Sciences

# Outline

- Conses, lists, true lists
- Basic functions on lists
- Some more macros
- The function rl
- ACL2s demo

# Quiz

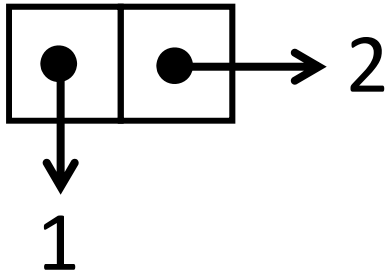
- How many of the conses below are expressions?
  A. 0
  B. More than 0



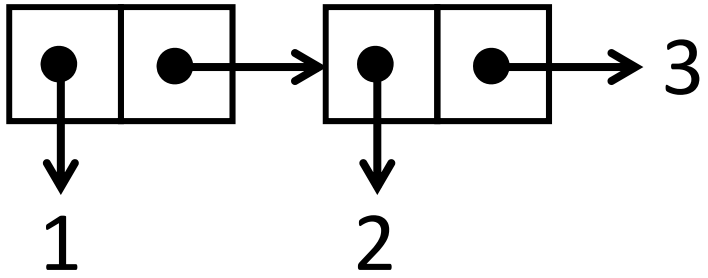
# Conses, lists, true lists

# Conses

- A cons is just a pair, e.g., (cons 1 2) :



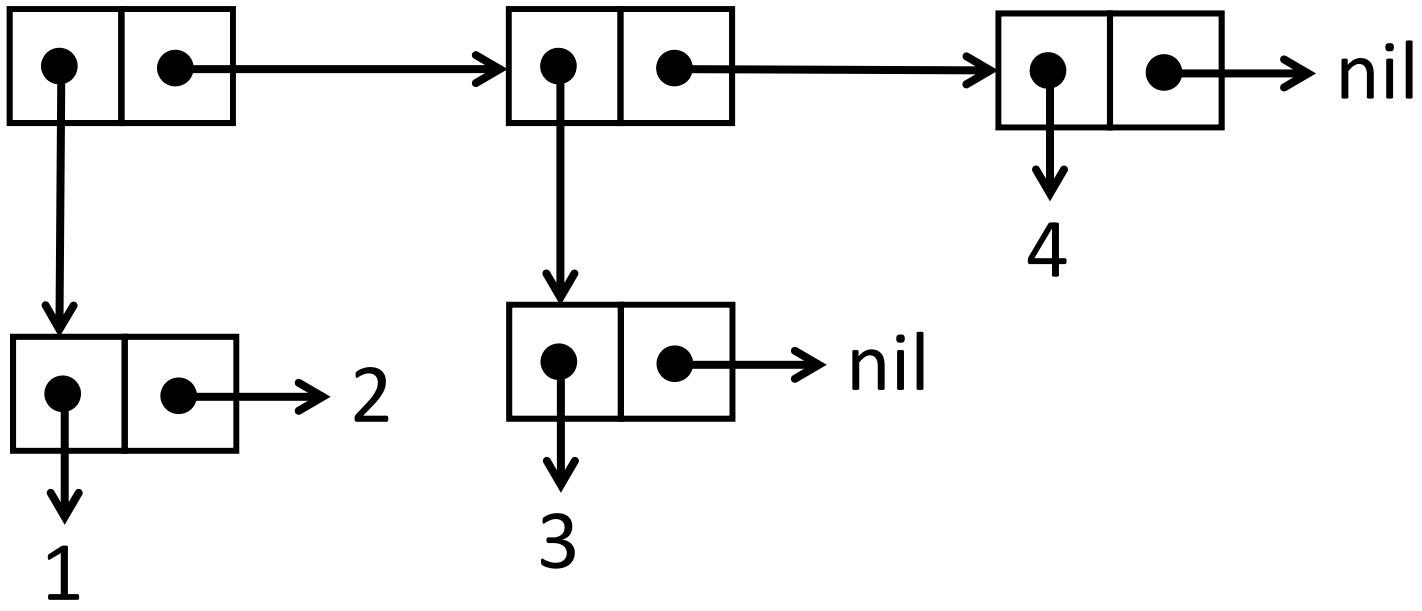

- Conses can be nested, forming trees:
  - (cons 1 (cons 2 3))




  - (cons (cons 1 2) (cons (cons 3 nil) (cons 4 nil)) ?

# Conses

— (cons (cons 1 2) (cons (cons 3 nil) (cons 4 nil))



# Semantics of cons: dotted pair notation

- ⟦(cons 1 2)⟧ = (1 . 2)
- ⟦(cons 1 nil)⟧ = (1 . nil)  (= (1), as we'll see next)
- ⟦(cons (cons 1 2) (cons (cons 3 nil) (cons 4 nil))⟧ =

= ( (1 . 2) . ( (3 . nil) . (4 . nil) ) )

# Conses: dotted pair notation

- **Simplification rules:**
  - (x . nil) simplifies to (x)
  - (… . (x …)) simplifies to (… x …)
  - Examples:
    - (1 . (2 . (3 . nil)))
    - ( (1 . 2) . ( (3 . nil) . (4 . nil) ) )

# Conses: dotted pair notation

- **Simplification rules:**
  - (x . nil) simplifies to (x)
  - (… . (x …)) simplifies to (… x …)
  - Examples:
    - (1 . (2 . (3 . nil))) -> (1 . (2 . (3)) -> (1 . (2 3)) -> (1 2 3)
    - ( (1 . 2) . ( (3 . nil) . (4 . nil) ) )
    - ( (1 . 2) . ( (3)  . (4) ) )
    - ( (1 . 2) . ( (3) 4 ) )
    - ( (1 . 2) (3) 4 )

# Quiz

- (1 2) is a:
  A. Expression
  B. Value
  C. Both an expression and a value

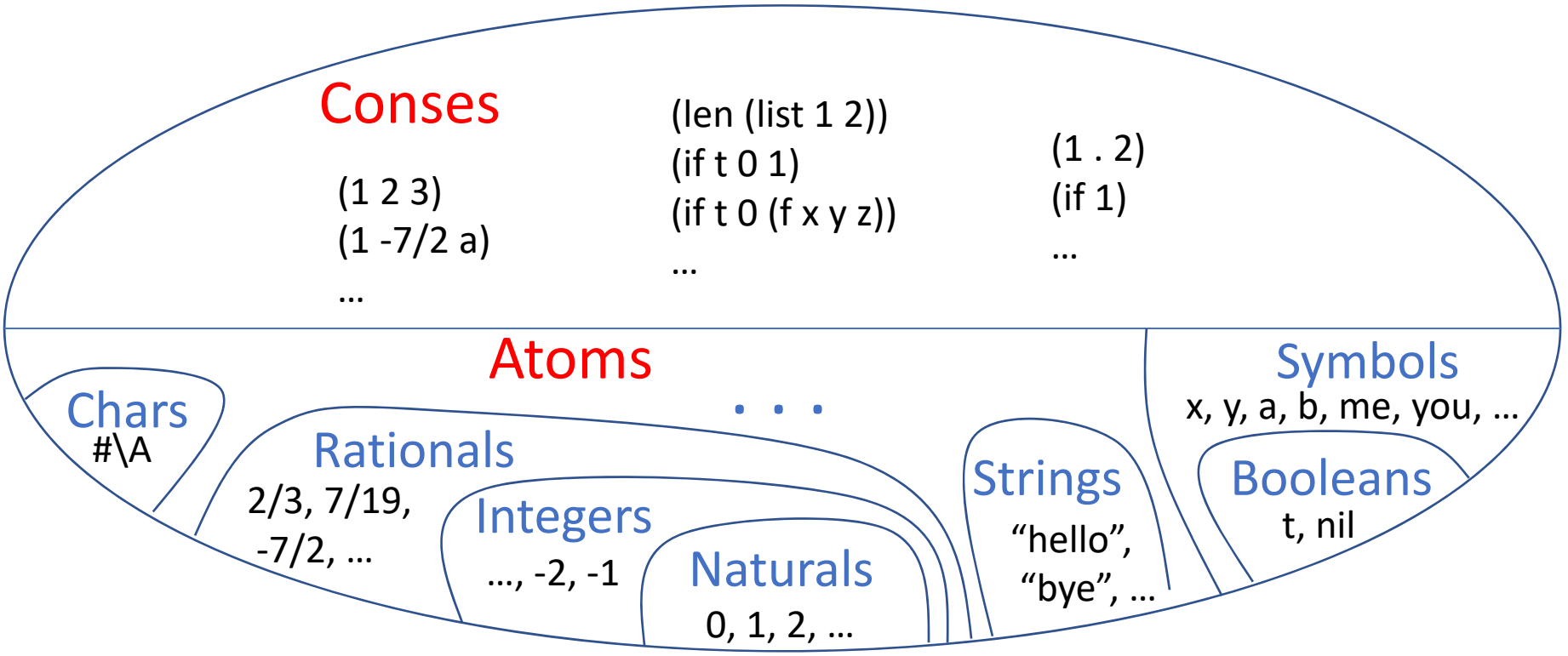
# Lists and true lists

Lists = Conses ∪ {nil}

True-Lists = $\bigcup_{i \in N} TL_i$ where: $TL_0 = \{\text{nil}\}$

$TL_{i+1} = TL_i \cup \{(\text{cons } x\ l) \mid x \in All, l \in TL_i\}$

Every true list is a list, but not conversely, e.g., (1 . 2).



# Quiz

- (1 2) is a:
  - A. symbol
  - B. atom
  - C. cons

# Quiz

- (1 2) is a:
  - A. list
  - B. true list
  - C. both

# Basic functions on conses

- Built-in functions:
  - `cons : All x All -> Cons`
  - `consp : All -> Boolean`
  - `car : List -> All`
  - `cdr : List -> All`
- Semantics:
  - $[\![(\text{cons } x\ y)]\!] = ([\![x]\!] \,.\, [\![y]\!])$
  - $[\![(\text{consp } x)]\!]$ is t iff $[\![x]\!]$ is of the form ( … ) but not (), i.e., not nil
  - $[\![(\text{car } x)]\!] = a$ when $[\![x]\!] = (a \,.\, b)$, nil otherwise
  - $[\![(\text{cdr } x)]\!] = b$ when $[\![x]\!] = (a \,.\, b)$, nil otherwise

# Basic functions on conses

- Semantics:
  - $[\![(\text{cons } x\ y)]\!] = ([\![x]\!] \,.\, [\![y]\!])$
  - $[\![(\text{consp } x)]\!]$ is t iff $[\![x]\!]$ is of the form ( … ) but not (), i.e., not nil
  - $[\![(\text{car } x)]\!] = a$ when $[\![x]\!] = (a \,.\, b)$, nil otherwise
  - $[\![(\text{cdr } x)]\!] = b$ when $[\![x]\!] = (a \,.\, b)$, nil otherwise

- Examples:
  - $[\![(\text{consp nil})]\!]$ =
  - $[\![(\text{car } ())]\!]$ =
  - $[\![(\text{cdr } ())]\!]$ =
  - $[\![(\text{consp } (\text{cons nil nil}))]\!]$ =
  - $[\![(\text{car } (\text{cdr } (\text{cons } (\text{if t } 3\ 4)\ (\text{cons } 1\ \text{nil})))))]\!]$ =

# Basic functions on conses

- Semantics:
  - $[\![(\text{cons } x\ y)]\!] = ([\![x]\!] \,.\, [\![y]\!])$
  - $[\![(\text{consp } x)]\!]$ is t iff $[\![x]\!]$ is of the form ( … ) but not (), i.e., not nil
  - $[\![(\text{car } x)]\!] = a$ when $[\![x]\!] = (a \,.\, b)$, nil otherwise
  - $[\![(\text{cdr } x)]\!] = b$ when $[\![x]\!] = (a \,.\, b)$, nil otherwise

- Examples:
  - $[\![(\text{consp nil})]\!]$ = nil (nil and () denote the same value)
  - $[\![(\text{car } ())]\!]$ = nil (car takes lists, not just conses)
  - $[\![(\text{cdr } ())]\!]$ = nil (idem for cdr, and ()=nil is a list)
  - $[\![(\text{consp } (\text{cons nil nil}))]\!]$ = t since $[\![(\text{cons nil nil})]\!]$=(nil . nil)
  - $[\![(\text{car } (\text{cdr } (\text{cons } (\text{if t } 3\ 4)\ (\text{cons } 1 \text{ nil}))))]\!]$ = 1 (work out why at home!)

# A recognizer for true lists

```
(definec true-listp (x :all) :bool
  (if (consp x)
      (true-listp (rest x))
      (equal x nil)))
```

# More macros

# Macros `first` and `rest`

- The macro `first` abbreviates `car`
- The macro `rest` abbreviates `cdr`
- Other macros:
  - `caar = (car (car ...))`
  - `cadr = (car (cdr ...))`
  - `cdar = (cdr (car ...))`
  - …
  - `second = cadr`
  - `third = caddr`
  - …

# The macro `list`

```
(list)         -> ()
(list x)       -> (cons x nil)
(list x y)     -> (cons x (cons y nil))
(list x y z)  -> (cons x (cons y (cons z nil)))
...
```

# The macro `cond`

The macro

```
(cond
  (c1 e1)
  (c2 e2)
   ...
  (cn en))
```

expands into

```
(if c1
    e1
    (if c2
        e2
        ...
                (if cn
                    en
                    nil) ... ))
```

We will always make sure to use as the last test `cn = t`, so that the final `nil` is never reachable!

# Next time

- Quote
- Let
- Data definitions